AFRL-RI-RS-TR-2015-067

EXPLORATION AND EVALUATION OF NANOMETER LOW-POWER MULTI-CORE VLSI COMPUTER ARCHITECTURES

OKLAHOMA STATE UNIVERSITY

MARCH 2015

FINAL TECHNICAL REPORT

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

FOR THE DIRECTOR:

/ S /	**/ S /**
THOMAS E. RENZ	MARK H. LINDERMAN
Work Unit Manager	Technical Advisor, Computing & Communications Division
	Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* MAR 2015	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* SEP 2011 – SEP 2014

4. TITLE AND SUBTITLE EXPLORATION AND EVALUATION OF NANOMETER LOW-POWER MULTI-CORE VLSI COMPUTER ARCHITECTURES	**5a. CONTRACT NUMBER** FA8750-11-2-0273
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) James E. Stine, Jr.	**5d. PROJECT NUMBER** T2SP
	5e. TASK NUMBER OK
	5f. WORK UNIT NUMBER ES
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Oklahoma State University, 401 Whitehurst Hall, Stillwater OK 74078-1030	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITB, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RI
	11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2015-067

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The research objectives of this work are placed on designing a complex Very Large Scale Integration (VLSI) multi-core architecture using an elaborate design flow or sequence of steps. Many of these architectures are currently or will be employed in advanced architectures that may have secure capabilities within the Air Force Research Laboratory in Rome, NY. This will be accomplished by designing complete design flow integration with commercial and open-source Electronic Design Automation tools. The design flow will take as inputs a high-level system-level architecture description, along with area, critical path delay, and power dissipation constraints. Based on the System on Chip architecture description and design constraints, the tools will automatically generate synthesizable Hardware Descriptive Language (HDL) models, embedded memories, and custom components to implement the specified VLSI architecture. Results show several orders of magnitude improvement over previous approaches with respect to designs for multi-core architectures, power dissipation strategies, and software reutilization.

15. SUBJECT TERMS

EDA Tool Flow, VLSI Multi-Core Design, SOC Design, HDL Models, Network on a Chip, Chip emulation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** THOMAS E. RENZ
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	29	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

The author is indebted to the talented engineers at the Air Force Research Laboratory (AFRL) who not only inspired the work presented here, but also encouraged the author with a rewarding and enriching experience. The author would like to thank many at the AFRL for their inspiration and kindness, including Jonathan Heiner, Pete Bronowicz, Dennis Fitzgerald, Steven Helmer, Andrea Lapiana, Giuseppe Lapiana, Rich Linderman, Tom Renz, John Rooks, Ross Thompson, Lisa Weyna, and Qing Wu.

1. INTRODUCTION

The overall goal of any computer architecture created in silicon is to facilitate an idea to a final design in an efficient and practical way. To accomplish this goal many designs are created through complex and elaborate software programs that write netlists or structural description of silicon structures by means of a concise software system or by producing a *design flow*. When engineers first started creating these silicon structures, the number of transistors or the integration of silicon devices that existed within a design was simple and straightforward. However, as the complexity of computer architectures increased over time, many of these software tools and design flows were demanding, elaborate, and extremely complex. Therefore, engineers resorted to creating and/or modifying software tools to help efficiently control the complexity of these implementations for the ultimate goal of producing Very Large Scale Integration (VLSI) computer architectures.

Although there are many open-source software tools and design flows to help create high-performance computer architecture designs, they seldom produce results that are on par with commercial VLSI software tools. This occurs because many VLSI software tools are produced by Electronic Design Automation (EDA) companies that have huge budgets that hire many programmers to tackle on-going research problems within computer architecture implementations. Although many publicly available components, standard cells, and high-level System on a Chip (SoC) descriptions are available for these VLSI tools, they are difficult to use due to their high amounts of complexity. This research aims at bridging the gap of what can be done with complex SoC descriptions of computer architectures combined with creating design flows targeting commercial EDA design tools.

Recently, a National Science Foundation panel (NSF) highlighted the urgent need that accurate modeling and evaluation of complex architectures to be a significant challenge for system architectures and digital system designers [1]. Ultimately, the NSF panel argues that simulation and benchmarking will require a significant leap in capability within the next few years to maintain ongoing innovations in computer systems and electronics. Even after several years since this seminal report illuminated what was needed in computer architectures, there are still huge gaps in what can be designed and budgeted for this task [2]. Consequently, there is a need for an efficient and reliable system that can be utilized for producing state-of-the-art computer architectures, especially for silicon implementations.

The research objectives of this work is to design, develop, and evaluate multi-core hardware support for computer architectures at the nanometer level. Many of these architectures are currently or will be employed in advanced architectures that may have secure capabilities within the Air Force Research Laboratory (AFRL) in Rome, NY. This will be accomplished by designing complete design flow integration with commercial and open-source EDA tools. The design flow will take as inputs a high-level system-level architecture description, along with area, critical path delay, and power dissipation constraints. Based on the SoC architecture description and design constraints, the tools will automatically generate synthesizable HDL models, embedded memories, and custom components to implement the specified VLSI architecture. It is anticipated that the results of this work will be a step closer to the guidelines outlined by the aforementioned NSF panel [1].

A key component of the design infrastructure will be that the tools will also generate simulation, synthesis, and place-and-route scripts and interfaces for the VLSI architecture, which can be used in conjunction with industry-standard design tools from Cadence Design Systems, Synopsys, and Mentor Graphics Corporation to obtain area, delay, and power estimates. Feedback from the design tools can then be used to modify the architecture description or design constraints, if necessary. An important part of these design tools is to evaluate methodologies to achieve low power designs and ensuring the design tools do not add malicious circuits and are secure.

2. BACKGROUND

Computer architectures are complicated by their fabrication creating complicated and elaborate vehicles to produce these silicon structures. During the 1970s and up through the 1990s, most computer architectures were created through vast layers of silicon deposited on a silicon substrate [1]. These depositions were usually fabricated in massive clean rooms that cost millions to build and maintain. However, as engineers progressed into the 21st century, computer architectures have also been able to be integrated through Field Programmable Gate Arrays (FPGAs). Although FPGAs are easy to design, and are far cheaper than most traditional computer architectures created through silicon, they consume significantly more area, delay and power [3]. Consequently, for computer architectures, which demand high amounts of performance, silicon high-performance systems are usually chosen.

The demand for increased speed, decreased energy consumption, improved memory utilization, and better compilers for processors has become paramount to the design of the next generation of computer architectures. To make matters worse, the traditional challenges of designing digital devices with semiconductor technology has drastically changed with the introduction of deep submicron technology. Designs that have been expanding Moore's Law have discovered that silicon technology has severe limitations within technologies below 180 nm [3]. What was once easy to improve a design by scaling the minimum feature size of a transistor can no longer be simply scaled.

Because silicon technologies are so small, designs can now implement billions of transistors on a reasonably small die. Unfortunately, this leads to power density and total power dissipation that is at the limits of what packaging, cooling, and other infrastructure can support [4]. More importantly, Complementary Metal Oxide Semiconductor (CMOS) technologies below 90nm have leakage current that almost matches or surpasses that of dynamic power, making power dissipation a major obstacle to designing complex SoC designs [3].

Although power dissipation complicates the process to which integrated circuits can be produced, it does not necessarily mean that designs cannot be efficiently designed. A designer just has to be cognizant that performance does not necessarily mean that one can increase the clock rate as technologies grow smaller. This new challenge requires designers to realize that both power and speed are closely linked and that engineering choices are normally required if a design requires a lower power factor and high clock rates [5].

To make things worse, processor designers have increased core counts to exploit Moore's Law [6]. This has made decisions about having multiple cores and the performance that it entails sometimes difficult to navigate. More importantly, single core processor designs are the engine that ultimately makes multiple core devices work. That is, for virtually all applications, including single-core general-purpose computer architectures, reducing the power consumed by SoCs is essential to allow new features that improve multiple core technology. Consequently, it is important to understand what and how power consumption affects SoC designs to improve upon it.

2.2 Power Dissipation

The total power consumption of a digital logic circuit consists of two major portions [6, 7]. The first part consists of dynamic power that is the power that is consumed when a device is active. Typically, dynamic power is consumed when devices are active and are switching back and forth. That is, they are based on what is supplied at the input of a circuit. For example, a circuit has lots of activity (e.g. within a router for the Internet), it will typically consume lots of dynamic power. Conversely, applications that only switch on during critical events (e.g. sensors within automobiles for abnormal events), they typically consume low amounts of dynamic power.

Dynamic power's main function is the amount of switching that occurs during an event [8]. Since most CMOS circuits are composed of layers of Silicon Dioxide, which is an excellent storage device, a majority of the switching power stems from the power that is charged and discharged from turning the transistor on and off, respectively. This typically results in a squared dependence on the voltage:

$$P_{dyn} = C_L \cdot V_{DD}{}^2 \cdot P_{trans} \cdot f_{clock} \tag{1}$$

where C_L is the load capacitance, V_{DD} is the supply voltage, f is the frequency of the system clock, and P_{trans} is the probability of an output transition.

In addition to switching power, internal power also contributes to dynamic power. Internal power occurs when a CMOS gate is suddenly turned from on to off and back to on. This switching causes both NMOS and PMOS transistors to be ON momentarily resulting in a short circuit or "crowbar" current. Although the short circuit can be small, it can contribute to the total dynamic power if the input is ramped up too quickly [9]. Short-circuit power can be described as:

$$P_{ss} = t_{sc} \cdot V_{DD} \cdot I_{peak} \cdot f_{clock} \tag{2}$$

where t_{sc} is the time duration of the short-circuit current and I_{peak} is the total internal switching current. Although short-circuit current will not be discussed in this report, it is important to make sure gates are not floating on an output when turning certain power-gating a circuit for lower-power consumption.

On the other hand, static power dissipation is defined as the power consumed when devices are powered up and no signals are changing values [8]. In the past, static power dissipation, which is mainly dominated by leakage current in a gate, was either non-existent or did not significantly impact a design. However, as the voltage and minimum feature size of a transistor gets smaller, the pronounced effect of leakage within a gate makes static power dissipation almost equal to or greater than dynamic power below 90nm [10].

In the past, traditional designs have resorted to lowering the power supply to get an exponential decrease in the power. This decision has been substantiated by Equation (1)'s power dissipation being dependent on the square of the supply voltage. The real problem is that lowering the supply voltage causes the drain to source current of a transistor to decrease. The drain to source current can be approximated by:

$$I_{DS} = \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot \frac{(V_{GS} - V_T)^2}{2} \tag{3}$$

where μ is the carrier mobility, C_{ox} is the gate capacitance, W and L are the dimensions of the transistor, V_T is the threshold voltage, and V_{GS} is the gate-source voltage. Since deep submicron technologies have low supply voltages, having a low threshold voltage allows CMOS designs to maintain good performance [8]. Unfortunately, as the threshold voltage gets smaller, an exponential increase in the sub-threshold leakage current (I_{SUB}) occurs.

The subthrehsold leakage current is the major dominant element within static power dissipation [6]. It occurs when a CMOS gate is not turned completely off. A good approximation to the subthreshold equation is shown in Eq. 4, where k is Boltman's constant, T is the temperature in Kelvin, q is the charge of an electron, and n is a function of the device fabrication process. The subthreshold leakage current for sub-90nm transistors is the major source of conflict within current technologies, such as the IBM cmos10sf 65nm technology used in this work. In the past, static power from leakage power was significantly lower than dynamic power, however, with newer technologies and shrinking power supplies, static power dissipation now is the dominant factor.

$$I_{SUB} = \mu \cdot C_{ox} \cdot \frac{k \cdot T}{q} \cdot \frac{W}{L} \cdot e^{\frac{(V_{GS} - V_T) \cdot q}{n \cdot k \cdot T}} \tag{4}$$

Equation (4) indicates that sub-threshold leakage, which is the predominant factor in static power dissipation, depends exponentially on the difference between VGS and VT. Therefore, as technology scales the power supply and V_T down to limit the dynamic power, leakage power grows exponentially, as was shown in [11]. To make matters worse, sub-threshold voltage current increases exponentially with temperature, which also complicates the process for low-power design.

Transistors are usually defined by their length and width, however, the former usually establishes the minimum feature size of a transistor [6]. As technology moves towards smaller feature sizes, the thickness of the oxide below the gate of a transistor also decreases in thickness. Unfortunately, in current semiconductor processes the thickness of the oxide is only several atoms thick. Consequently, the thinness of the oxide establishes a current that tunnels through the gate towards the channel of a transistor, so much so that in current processes gate leakage can be nearly *1/3* as much as sub-threshold leakage [7]. In order to reduce the gate leakage, some manufacturers have resorted to high-K dielectric materials, such as Hafnium, to keep the gate leakage in check [11].

Another technique to reduce the leakage current is to use multi threshold voltage transistors. Using this technique, high V_T cells can be utilized wherever performance goals allow the power dissipation to keep in check. Specifically, having transistors that can utilize different threshold Voltages, usually associated with Multi-Threshold CMOS (MTCMOS) circuits, allows the reduction of the substrate current, as shown in Equation (4). And, lower V_T cells can be used on a critical path to meet a specific timing, because lower threshold voltages have faster propagation delays as they switch faster [9]. The technology utilized for this work is IBM's cmos10lpe 65nm

[12] and cmos32soi 32nm [13]. Both technologies enable the use of regular V_T, high V_T, and low V_T standard-cell transistors to reduce the gate leakage and improve speed gains on the critical path.

2.2 System on Chip Design Flow

Many design flows involve taking structural or behavioral descriptions of computer architectures and translating them into a working silicon mask layer that can be fabricated. Although this process is just an evolution what software compilers use, this process has dramatically changed from early designs involving several hundred transistors to current System on Chip designs that encompass close to or exceed 1 billion transistors [14]. To make matters worse, power and high-performance issues have complicated the entire process [5].

Standard-cell designs involve taking pre-made layout elements, such as an AND or NAND gate, and having software stitch elements together via placing each layout and routing wire between known pins. Early layout editors, such as the Magic Layout Editor, had built-in routers to allow designers to avoid having to worry about laying out wire between two points [15]. However, as more points were created between a pin and the cost for a given route increased, there was a dramatic need for more algorithms to deal with congestion and efficiency [16].

Software has been written to translate or parse high-level descriptions of digital systems into a representation that allows hardware to optimize and map to a standard-cell library. More importantly, many of these points that are parsed and subsequently lexed within a software tool can be connected from standard-cell parts to another standard-cell part, custom-cell part or pin. Therefore, it is important that software can translate, optimize and map a high-level description into these netlists accurately and concisely. Typically, this process of translating, optimizing, and mapping is called synthesis [6].

After synthesis, netlists can be utilized to place standard-cells, custom-cells, input/output pins and drivers, memories, and other ancillary parts onto a grid. This design will be optimized for placement by its wire-length, power connections, and other elements of cost associating with each tool. Consequently, the process from going from idea to final mask layer for silicon fabrication can be broken into two distinct phases: front-end processing and back-end processing [17].

Another important concept is that many front-end and back-end tools utilize heuristics to accomplish their algorithm. That is, they tend to have NP-hard (i.e., non-deterministic polynomial-based in solving) [17]. Therefore, each time an algorithm runs it may result in a different outcome, yet close to an optimal answer [6]. This was one of the main reasons this research incorporates tools from professional EDA vendors in that they can produce the best outcome giving a set of high-level netlists and constraints.

The front-end usually is associated with synthesis and any preliminary placement of parts that have been subsequently mapped during the synthesis process. Some tools have been able been able to pre-place parts to aid in the synthesis process (e.g., through topologically mapping in Synopsys Design Compiler), however, most flows usually start the front-end process by first synthesizing and then placing parts initially onto a grid.

The grid is important in that it allows all the pins to connect together and a well-defined grid helps the front-end get its job done quickly and accurately [17]. It also simplifies the process in figuring out wire length, which is crucial to many constraint-driven EDA tools [16]. A grid that is chosen that is too big or not large enough may result in an objective that does not meet a cost criterion or worse yet, a placement that cannot connect all pins for a given netlist. To help designers, most technology kits that come from commercial fabrication sites choose the grid for their users. In this paper, the technology kit comes from IBM and are all drawn at 5 nm.

The back-end involves the numeric crunching that occurs once an initial placement of parts is set to a grid. During the back-end process the software tools typically move some of the placement around and finally place & route the pins together. Each given design has a constraint for a given objective, whether its power dissipation, energy consumption, and fast critical paths. The back-end may also report timing and power/energy reports that help users accurately report on a given design.

3.0 METHODS, ASSUMPTIONS AND PROCEEDURES

The goal of this paper is to research and develop techniques, tools, and flows for high-level synthesis of SoC platforms in deep sub-micron CMOS technologies that (1) provide the ability to efficiently integrate embedded memories, processors, hardware accelerators, and communication structures, (2) utilize synthesis and layout information to accurately estimate area, delay, and power from high-level SoC architecture descriptions, (3) facilitate design-space exploration and component reuse in multiple core SoC solutions, and (4) are well documented, easy to use. This goal will be accomplished by researching and developing high-level design flows for complete SoC solutions and using computer tools to explore new techniques for creating fast critical paths and exploiting power management.

The high-level synthesis tools will take as inputs a high-level SoC architecture description, a parameterized library of configurable SoC components, and design constraints. The tools use these inputs to generate synthesizable HDL models, embedded memories, and custom components to implement the specified SOC architecture. The tools also generate simulation, synthesis, and place-and-route scripts for the SoC architecture, which are used in conjunction with industry-standard design tools. A major element of the work produced in this research is that a variety of commercial tools can be used together or separately. To accomplish this feat, specific interfaces are created that allow many of the tools to exchange information together.

In addition to the design flows and tools, this research produced hardware accelerators, functional units, processors, memories, and communication structures for use in low-power SoC systems, such as multimedia PDAs and digital cameras. These components are characterized in terms of area, delay, and power dissipation and used in conjunction with a flexible simulation framework to facilitate rapid design space exploration of new SoC solutions and power management techniques. Power efficiency is targeted at the system, architecture, circuit, and layout levels to provide a firm framework for the design and evaluation of future applications.

The following elements were developed for this project at the Air Force Research Laboratory:

- Design flows and SoC components for integration into a complete System on Chip design for multiple commercial EDA VLSI tools.
- Create extensible test environments that allow for easy chip exploration and analysis
- Develop multiple core relaxed consistency memory architecture for use within possible AFRL secures processor design architectures.
- Each of the subtasks is described in the following subsections. As summarized above, the subtasks together focus on the development of high-level EDA tools for low-power SoC designs.

Although some of the items have been previously implemented, one of the major elements to this work is the integration of components for rapidly smaller transistor sizes. As transistors get smaller and smaller, the major element that impedes designs is wiring or interconnect [6]. Consequently, as more and more elements are put together on a device, electrical effects such as current drive are important. For example, a wire that only traversed a small distance in previous designs may in fact have several microns to travel for larger System on Chip (SoC) designs.

Therefore, wires may actually not pass correct digital voltages across a wire without proper examination. These effects are typically called Signal Integrity issues.

Additionally, one of the deliverables for the work performed during the this work was to develop Very Large Scale Integration (VLSI) design flows and tools, researched and develop highly optimized standard cell libraries, functional units, processors, memories, and communication structures for use in low-power small feature size SoC systems. These scripts are put together in order to make sure Signal Integrity issues as well as variations within semiconductor devices are integrated together with the scripts. In many cases, the scripts are documented with easy to learn elements that help understand how the tools combat problems such as signal degradation, variation of timing effects, and other anomalies caused by small transistor size components.

3.1 System on Chip Framework

This framework consists of an integrated set of design tools and flows that take a high-level SoC architecture description, a parameterized library of configurable SoC components, and design constraints. A high-level architecture description can be utilized using a combination of Verilog and VHDL Hardware Descriptive Languages (HDL) and a parameterized description of the components used to implement the system. For example, when specifying a configurable processor the description might contain information on the datapath size, the number and type of functional units, and the subset of available instructions supported. The tools then use these inputs to generate synthesizable HDL models, embedded memories, and custom components to implement a specified SoC architecture. The initial target application of the work performed in this work is designated for the IBM cmos10lpe *65*nm technology, however, the tools were designed so that they could easily be translated to other technologies. Specifically, the tools are also targeted for IBM cmos32soi *32*nm technology, as well.

With the advancement of many technologies that are involved in Electronic Design Automation (EDA) software, the process of creating high performance digital systems becomes possible. However, the importance of creating a repeatable engineering process involves a delicate and balanced approach to engineering design and utilization of this EDA software. Several companies have been integral in creating this process including Cadence Design System (CDS) ®.

Cadence Design Systems® was one of the first EDA companies that specialized in designing silicon systems that have designs utilizing multiple methodologies using their First Encounter tool. Using this approach, modern designs can be comprised of custom-based silicon devices, designs created with Hardware Descriptive Languages (HDLs), as well as Intellectual Property from third party companies. To give the tool more symbolism at its power of pulling together multiple digital systems, CDS now calls its tool Encounter Digital Implementation™ (EDI). That is, EDI’s overall purpose is providing an EDA tool for the assembly of Systems on Chip (SoC) designs. Although EDI works efficiently its use is predicated on commands or instructions that users give EDI the ability to integrate each of these layers. Unfortunately, as time has progressed EDI has had difficulty in maintaining commands that are unified and cogent for users to accomplish their intended purpose.

To make EDI more understandable and easy to use, CDS created a sequence of commands or a design flow that is integrated as a template for use with their tool. These scripts are called the

Encounter Foundation Flow (EDF). The EDF allows users to easily get EDI working by having users focus on what files they need instead of the command they need to use. Although many designs still require users to optimize many of their commands, the EDF gets users started and allows them to focus on using the tool efficiently. An example design is shown in Figure 1 using 65nm IBM technology and the EDF:

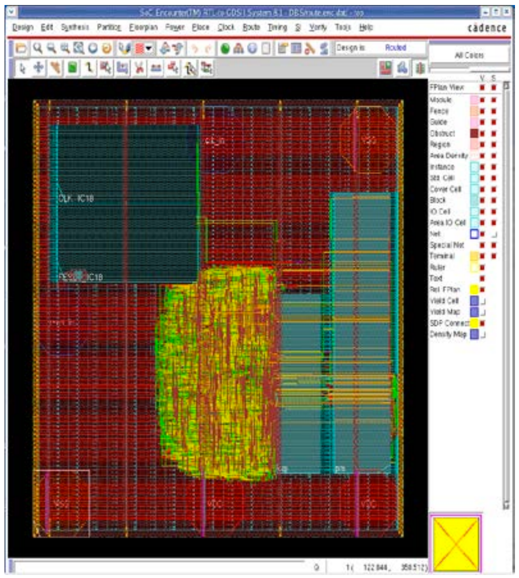

Figure 1. Sample Cadence Design Systems Encounter Tool Screenshot

To help the design flow integration and EDF work together, a set of Makefile scripts are created to allow all the tools to run independently. More importantly, the Makefile scripts also allow log files, report creation, and storage of intermediate files that allow designs to be preloaded or stored for later use. Makefiles are important in computer-based tools in that they are text files written in a certain prescribed syntax to allow software to organize code, its compilation, and production of subsequent compilation. The Makefile structures uses the following syntax for each compilation step:

1. init : initialization of chip and reading of required input files
2. place : placement of chip design after pre-placement. Pre-placement is done manually to allow the chip to give a head start on what parts are better to be placed together or next to each other.
3. pre-cts : initial clock-tree synthesis and examination of the clock tree.
4. cts : clock-tree synthesis to optimize clock-tree
5. post-cts : examination of the automatic clock-tree synthesis and whether it meets specific chip requirements.
6. route : routing of netlist using wires based on cycle-time optimization and/or power-based considerations.
7. postroute : examination of cycle time to see min-time constraints (i.e. hold-time).
8. final : final-chip connections and report creation.

Each stage of the Makefile sequence corresponds to a stage or sequence in the front-end or back-end process. The design flow sequence is shown in Figure 2. All elements within the Makefile are designed to allow users to go between stages. That is, a user could theoretically run "init" and then stop. The Makefile creates specific flow outputs that are utilized for the EDA tools in creating reports, intermediate files, and any ancillary file helpful in the design flow.

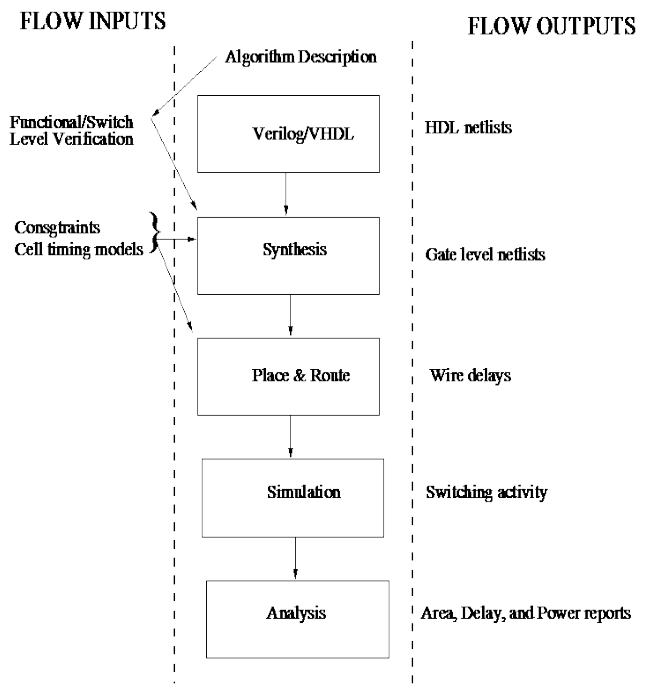

Figure 2. Design Flow Structure for System on Chip Framework

In addition to a Makefile, many of the scripts in the design flow are created within a script-based language that most EDA-vendors support called Tool Command Language (Tcl). John Ousterhout invented Tcl while he was a faculty within the University of Berkeley mainly for a public domain VLSI tool called Magic [15]. The Tcl language is useful in that it has an easy-to-learn format and also works well with Graphical User Interfaces (GUIs) from most EDA tools. Moreover, Tcl is now accepted as a widely supported format for most EDA tools.

Initially, all scripts were created with CDS' EDI tool in mind. This was done as a matter of convenience, but the ultimate goal is to have the best EDA tools related to System on Chip design integrate. Currently, there are two popular commercial EDA tools that support SoC design: Cadence Design Systems® (CDS) Encounter Digital Implementation™ (EDI) and Synopsys® IC Compiler™ (ICC). Both tools are important leaders in the SoC filed in that EDI tends to be used more for optimizing cycle times, whereas, ICC tends to be used for low-power options.

An important part in creating this research is allowing the EDA tools to work together. An important impediment in the construction of the SoC framework is that EDI and ICC use different file formats. EDI uses a format based on something called the Layout Exchange Format (LEF). Although CDS has moved more towards a new format called Open Access (OA), this research utilized formats based on the LEF file structure. This was done, because LEF is a format that is common in the VLSI community and CDS has indicated it will continue to support LEF into the future. On the other hand, Synopsys ICC uses something called Milkyway formats. Therefore, it was important to create scripts that allow many of the tools to work together.

To help with the creation of a digital design, a common_setup Tcl file is utilized to allow users to add information for the IBM *65*nm and *32*nm technologies. This Tcl file has information related to core Synopsys file geometry and area/delay/power information. To allow a design to work within Synopsys® ICC, a common file structure or database is utilized called MilkyWay. Before a design could be used with Synopsys ICC, the Milkway database was created using the command:

milkyway –galaxy –nogui –tcl –log memory.log one.tcl

As stated previously, it is important to create tools that work together. Custom-based Makefiles are created to create the same structure that was exhibited within the EDI tool system including the naming structure (e.g., init, place, route).   The Makefiles are integrated into one structure and allow either tool to flow independently from each other.

EDA tools are defined to work with different file structures. Although common file formats are available for some EDA tools (e.g., OpenAccess), most EDA systems tend to use defined file formats that are unique to a given tool. This allows companies to tie commercial tools to a given company. That is, if a company devotes 2-5 years into a SoC design and it uses a proprietary format from CDS, it is less likely to change EDA tools. Typically, Synopsys® tools use Milkway databases, whereas, Cadence Design System® use Layout Exchange Format (LEF) formats. To help create tools that allow common exchanging of data, Tcl scripts were modified to output the creation of Design Exchange Formation (DEF) files.

The Design Exchange Format (DEF) format is an open specification that represents physical information of a design, sometimes called a layout, in an American Standard Code for Information Exchange (ASCII) format. As opposed to LEF files that only have geometry information on certain layers, DEF files contain more information on each underlying structure. Unfortunately, DEF files usually only contain netlists, component placements and its routing information. The geometry of each individually placed element or the individualy items within each SoC, either LEF or Milkyway is still required. Therefore, it was important that each tool output only DEF, but that it still retains the structure of its geometry components (i.e., LEF and Milkyway).

In order to get the scripts to work together, the scripts were modified to write out DEF information at each independent stage. DEF conveys the logic design data as well as the physical information to either EDI or ICC. This physical information includes the physical placement locations, orientations, routing geometry data, as well as the logic design changes for back annotation of the design to any netlist. Although the DEF works well in converting files between multiple EDA tools, it does not help with the setup information. Consequently, it is important that the setup for both Synopsys and Cadence be utilized for any integration and the tools within this work.

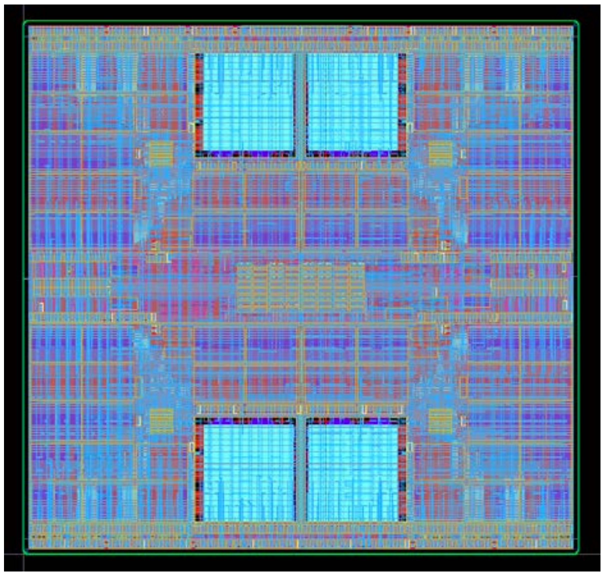
Figure 3. 4 Core Secure Processor Layout using System on Chip Framework EDA tools

Another important element is that the DEF file does not contain the file format that each tool utilizes. Therefore, it is important that if a design is to be translated between two tools (e.g. EDI → ICC), it must have an initial file format to work with. Therefore, a design should have an initial floorplan to work with, so that it can import the DEF correctly. Otherwise, the file formats may not align properly. This can easily be achieved by initializing the place & route process within EDI or ICC (e.g., the init phase) and then importing a layout from DEF.

During the course of this research, one particular design was taped out for fabrication using both the Cadence Design Systems® and Synopsys® tools. The current design involves four cores that involve secure processors. Each processor has a set of memories, drivers, input/output pins, First-in First-Out (FIFO) buffers, and standard-cells. Figure 3 shows the final layout before fabrication. The design was optimized for 300 MHz and a low power dissipation footprint.

3.2 System on Chip Test Chip Environment

This work also integrated a low-power standard cell library flow for static CMOS logic, configurable Random Access Memory (RAM), Read Only Memory (ROM), pad and input/output drivers, and cache memory generators using IBM and Virage Logic Design Intellectual Property designs. The SoC components are integrated into the high-level synthesis framework to facilitate the design of complex SoC solutions. In developing these components, one goal is to utilize the design framework to generate different functional units and processors to evaluate potential tradeoffs in key design decisions. For example, adding new instructions and functional units might help improve the performance of certain 3D graphics applications, yet cause an unacceptable increase in static power and area. Having a design framework that can quickly generate and evaluate a wide variety of designs will assist in making correct decisions early in the design process.

A layout for a SoC design can normally be simulated including its underlying parasitics. This information is invaluable to this project since decreasing feature sizes and increasing chip sizes make the influence of wiring parasitics on the performance of the chip critical or even dominant. Therefore, as noted by International Technology Research (ITRS) roadmap, gate delays may decrease, but wiring delays may increase as minimum feature sizes decrease [6]. In order to guarantee accurate simulations, two-dimensional and three-dimensional extractions were implemented within all commercial tools, such as Cadence Design Systems' Fire&Ice and Mentor Graphics' Calibre, for the proposed SoC design tools and framework. Having detailed extraction is paramount since existing design flows with this capability are minimal or non-existent. Consequently, each design can then be fully characterized in terms of area, delay, and power dissipation and utilized in conjunction with a flexible simulation framework to facilitate rapid design space exploration of new SoC solutions and power management techniques. Figure 4 shows sample design of the testbed.

The design flow is then integrated along with the parameters cells provided by IBM. Cadence Design System's Virtuoso is used for layout creation and schematic entry, however, the design flow is designed to work with Synopsys and Cadence Design System synthesis engines. Cadence Design Systems' EDI and Synopsys' ICC are utilized for place & route, and Mentor Graphics' Calibre is used for Design Rule Checking, Electrical Rule Checking, Layout Versus Schematic, and Parasitic Extraction (PEX). The PEX is extremely important for power dissipation estimation, since capacitance in the wire is required to adequately address the total static and dynamic power dissipation within a design.

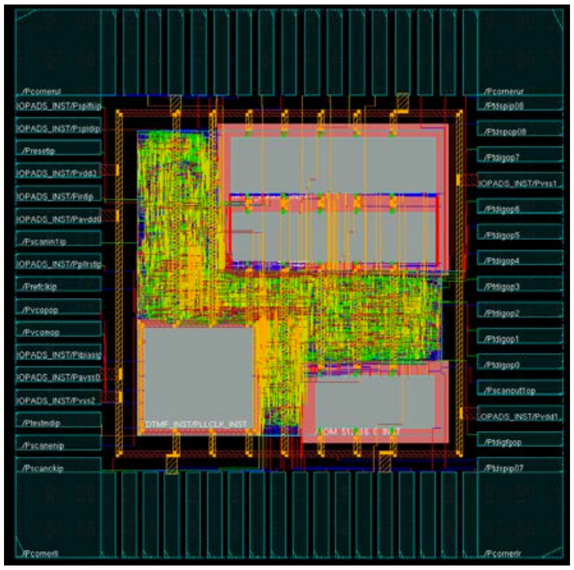

Figure 4. Sample System on Chip Testbed for Testing and Innovation

3.3 Computer Architecture Simulation and Memory Coherence/Consistency Modeling Environment

Computer organization is typically limited by the speed of its underlying structure. For example, a piece of silicon has a certain delay, area, and power property that is contingent on how it is drawn and organized when manufactured. For most computer architecture's this has been a challenge, because most parts within a computer architecture have different organizations and internal structures. Typically, a designer has to balance these circuit dependencies within the technology and algorithmic optimizations for a given computer architecture. One such challenge is the process of designing logic within computer architecture to process data, such as with carry-propagate adders, and its interaction with storing this data.

Most computer systems store this data within pieces of logic that allow data to be read and written as needed. These "memories" are crucial to any computer system and in some ways really define a computer system. In other words, a computer system without some storage mechanism, such as memory, is not really definitively named. Although memories have revolutionized computer systems they tend to have problems synchronizing with the computation part of any computer system, typically called its datapath. This occurs, because any memory system usually relies on some form of feedback that consumes a given amount of time to resolve [17]. Since memories are typically slower than most datapaths, they tend to be the defining computer element for speed and solving problems efficiently.

Unfortunately, the gap between memory performance and that of the datapath has grown quite considerably over the last twenty years [18]. Although computer designs have designed new ways to create better memory architectures they still are outpaced for every datapath design created today. To compensate for the growing "memory gap", designers have hidden this lagging speed within an increase in the processor speed [6]. To aid in compensating speed for performance, memory designers have also targeted multiple memory hierarchies to help faster memory accesses occur more frequently. That is, memory with faster speeds but smaller sizes are stored closer to the datapath, whereas, memory with slower speeds and larger sizes are located further aware from the datapath. Digital logic aids the transition from hierarchy to hierarchy within the memory in the form of something called a "cache". Much to the dismay of most computer designers, the speed at which computers can operate is reaching a theoretical limit imposed by the underlying silicon structure of the processor.

To compensate for the limited speed at which computer systems can operate at computer designers have branched out in increasing the performance of systems in parallel. In other words, instead of worrying about a given Instruction per Cycle (IPC) performance, designers have worried about Threads Level Parallelism (TLP). A thread has many definitions, but in its purest definition it can be labeled as a baseline operation for a given processor [17]. To help processors shift from IPC to TLP, processors have difficult tasks in synchronizing memory within multiple hierarchies. For example, if a computing system has multiple cores, it will typically have multiple caches and design logic within a computer architecture must make sure these caches are synchronized across multiple cores.

Most computer designers identify the problem of synchronizing memory across memory hierarchies into two separate classifications: coherence and consistency. Coherence defines what values can be returned by a read and Consistency determines when a written value will be returned by a read [19]. Coherence and Consistency are complementary in that coherence defines the behavior of reads and writes to the same memory location, whereas, consistency defines the behavior of reads and writes with respect to accesses [19]. This basically boils down to making sure reads and writes are not going to get interrupted or happen before each other. In other words, memory is written and read such that all operations are atomic or can be done in such a way that no intervening operation can occur.

Traditionally, coherence has been handled by local digital logic that tracks the state of the cache in which it identifies. The earliest textbooks identifies these algorithms as adaptive Finite State Machines (FSMs) that continuously "snoop" the bus and watch for any addresses on the bus are within their caches. If a corresponding cache is identified, the data in the cache is either "invalidated" or removed, otherwise operations progress as normal. As one can imagine, every bus transaction must check the cache address tag and potentially interference with processor cache accesses. This problem is exponentially magnified as the number of cores is increased, since all memory operations have to be broadcasted to all other cores. Therefore, as the number of cores increase, the local traffic scales and reflects a significant impact upon any processor basic metrics.

Similar to digital logic, keeping track of memory accesses can be limited within the separation of data. That is, shifting the memory by distributing the memory for local memory traffic and from other areas. That is, make a hierarchical system of memory tracking similar to how carry-lookahead adders distribute carries across a network [17]. To accomplish this task it is sometimes easier to not use a Finite State Machine snooping protocol and distribute the memory within a table using a directory protocol [19]. This directory keeps the state of every block that may or may not be cached as well as which caches or collection of caches have copies of the block, whether it is dirty, and so on. They key is to distribute the directory so that the coherence protocol knows where to find the directory information for any cached block of memory. Subsequently, distributing the directory also enables different coherence requests to go to different directories.

As with any commercial system, the problem for most designers is that there is nothing available to help researchers develop these methods. One of the goals in this work is to integrate the ideas into the current secure processor multiple core design that the Air Force Research Laboratory is currently utilizing. Therefore, to help implement the design, a Verilog implementation was researched and a directory protocol was integrated to help in coherence. The problem is also compounded with the fact that any process or heavyweight thread could potentially be composed of more threads within each process. This is typically called a lightweight process and makes the problem of multithreading more difficult, because a single program can be made up of a number of different concurrent activities. This make the problem of consistency within current computer architectures difficult to describe and even more difficult to handle coherence. The problem is really broken down into what properties must be enforced among reads and writes to different locations by different processors.

To help this process, researchers have considered using the directory structure and relaxing to make sure all reads and writes complete and are synchronized by their ordering. That is, operations for read and write are atomic. This is typically called a *relaxed consistency* model and the directory structure considered for this work comprises a relaxed method for ordering the writes after reads, reads after writes, writes after writes, and finally reads after reads.

As stated previously, it is difficult to implement systems that have complex multiprocessor behavior and also contain multiple cores. Therefore, a pipelined Microprocessor without

Interlocked Pipeline Stages (MIPS) model is utilized that is well studied and has been already designed by the author in Verilog [18]. The Verilog model contains 4 cores and is fitted with the Virage Memory architecture that the current AFRL secure processor utilizes. Each core has its own cache, but the L2 cache has a directory structure locally to keep track of all reads and writes within the processor. Each directory structure that implements the coherence protocol is simplified and is also based on similar architectures by the Sun Niagara T2 architecture that is available on the Internet [20]. The basic architecture is shown in Figure 5. The directory is integrated within the L2 cache similar to the Sun Niagara T2 and each directory has an arbiter that allows multiple elements to communicate with the other cores distributively.

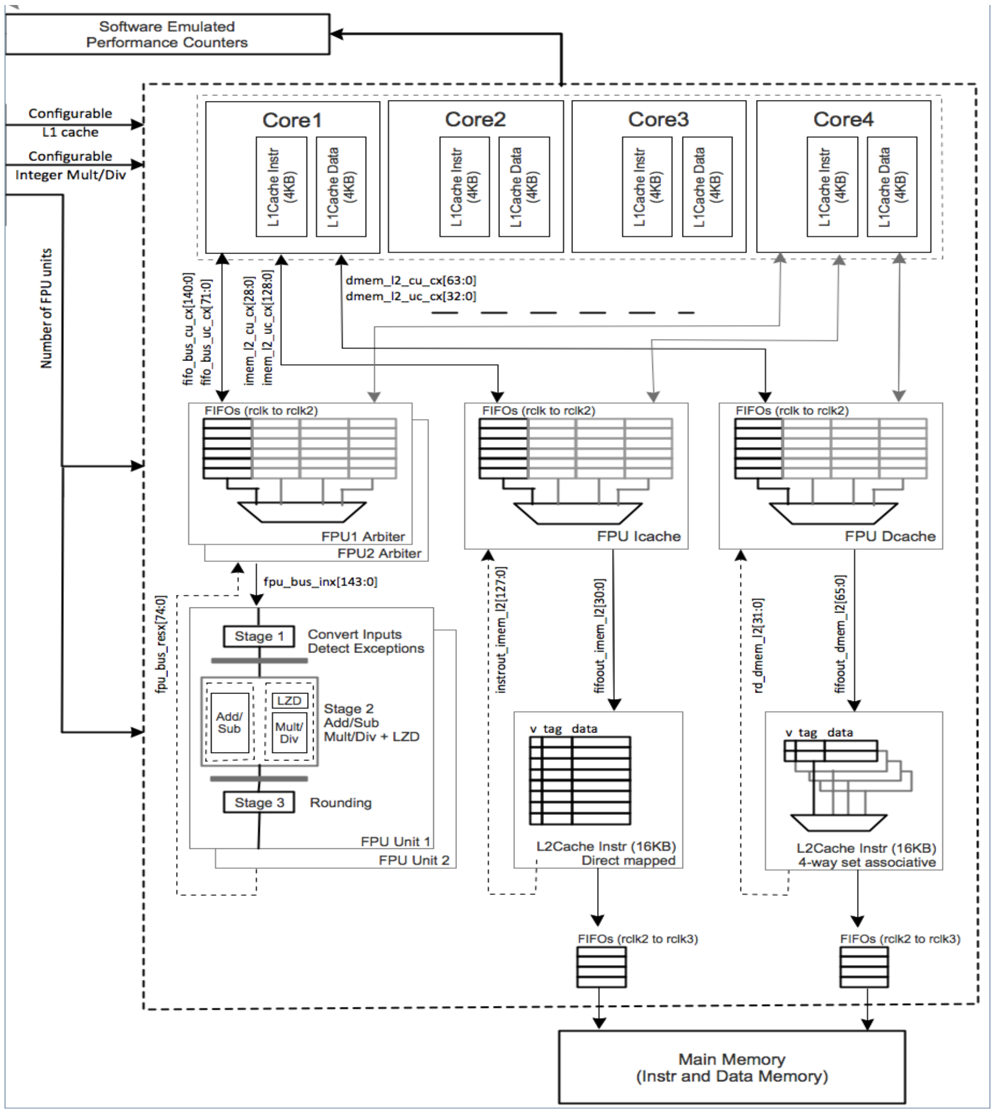

Figure 5. Basic MIPS MultiCore Directory Architecture

The Verilog model is extensively verified using several sample programs and testbenches. All testbenches verified that the coherence and consistency protocols work well within the context of the Verilog implementation. The Verilog is also written using Register Transfer level (RTL) - level constructs that can be further synthesized and placed & routed using the SoC design framework discussed in Section IIB. An important part of the Verilog model is that it incorporates a simplified MIPS model. This allows a user to study the design well, since the MIPS is well understood and a basis for most modern computer architectures. Consequently, once a user can understand a given protocol for the directory structure, it is anticipated that it can be used in future secure architectures within the AFRL that might use shared memory accesses.

4.0 RESULTS

The design flow is created in such a way that environmental variables are utilized to enable it to be installed anywhere. For example, all the libraries refer to an environmental variable within the common_setup Tcl file discussed earlier. The IBM cmos10lpe 65nm SoC framework utilizes standard-cells and memories from Synopsys® (formerly Virage Logic). On the other hand, the IBM cmos32soi *32*nm SoC framework uses standard cells from ARM and memories from IBM. Both implementations utilize MTCMOS-based standard-cell technologies. Although the standard-cell implementations have been extensively tested, the memories are still not completed tested for IBM's cmos32soi *32*nm technology. This is because they were not available for the context of this work from IBM. However, they can easily be inserted by modifying the common_setup Tcl file once the memory becomes available.

Scripts are designed to work with the design flow, so that any design can be easily created from a HDL design into a mask layout. All standard-cell designs and memory elements are created using scripts that either generate a placed and route design or initiate a language within the Cadence Design System® or Synopsys® EDA tools to create a memory array.

Power dissipation is an important element within any computer architecture, however, the computation of power can be complicated by the level of extraction that occurs for a given design. Consequently, the SoC framework and design flow presented here has several different power level estimation tools that can either estimate the power during synthesis with no parasitic extraction of the wires (e.g. through Synopsys® PowerCompiler™ or PrimeTime™) or computed more accurately using an extracted SPEF file (e.g. through Synopsys® nanosim™). Scripts are crated that allow all designs to be tested effortlessly for timing, area, and power parameters. More importantly, all the scripts can be modified, so that power, delay, or area can be targeted as an optimized constraint for a given computer architecture. All designs were created and designed effortlessly through the use of scripts designed to interface the HDL definitions.

The designs presented in this research are designed to allow a user to focus on a particular item for optimization, such as Signal Integrity, and play with design parameters to enhance its understanding and potentially go between different EDA tools. Right now, only Synopsys® ICC and Cadence's® EDI are utilized, but this could be expanded to other tools, if needed. The designs are all self-contained with the directory and can be remade through their Makefiles or by typing, "make all". Since each design is composed of smaller implementation design architectures, each design runs with minimum run time and system resources. After each step is run, a DEF file is saved at the end to allow conversion between each tool.

All of the designs are created with the cutting-edge and most-recent EDA software provided by EDA tool vendors. That is, the process of going from design architecture to deployable silicon, or back-end process, is utilized from Cadence Design Systems and Synopsys separately and self-contained within one set of scripts. Cadence Design Systems® or CDS utilizes Encounter Digital Implementation™ and Synopsys® employs IC Compiler™. That is, a full set of scripts from both manufacturers has been created for this work in hopes that they can be explored and utilized separately or combining them. Since the front-end process or the synthesis portion of the design flow is common to both, both systems employ Synopsys® Design_Compiler™ or DC for turning the SoC architecture into an implementation based on a technology-mapped netlist (i.e., the front-end process). Moreover, the simulation tool ModelSim from Mentor Graphics is also employed in each design for HDL simulation and verification, since it is a rich and full featured to contain most ideas within simulation.

All designs are contained on the Oklahoma State University/AFRL research server. It is also anticipated that many Air Force Research Laboratory personnel can utilize the research presented in this report for further study and other projects.

5.0 CONCLUSIONS

This paper demonstrates the research that was conducted for the Air Force Research Laboratory. With the combination of architecture and circuit-based enhancements and a common set of software design flows, allowing a cohesive SoC framework that produces designs that are several orders of magnitude better than implementing them manually or without the framework. More importantly, the tools have been created that allow them to create multiple design seamlessly as well as create an agile environment for the deployment of VLSI computer architectures in silicon.

The design flow is designed to work out-of-the-box and allow anyone to design fast and efficient designs with the IBM cmos10sf *65*nm or IBM cmos32soi *32*nm technologies. There is still significant additional work that can be done by incorporating better variation control within a design, power gating, clock gating, and utilizing Synopsys® Unified Power Format language. This language is useful in allowing many designs to optimize energy and power consumption within mask layouts. Future work will be ongoing in this area and produce better and more productive computer architectures with multiple-cores and better security enhancements.

REFERENCES

[1] K. Skadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai, "Challenges in computer architecture evaluation," *Computer*, vol. 36, pp. 30-36, 2003.

[2] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley*, et al.*, "Rethinking Digital Design: Why Design Must Change," *Micro, IEEE*, vol. 30, pp. 9-24, 2010.

[3] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed.: Addison-Wesley, 2010.

[4] M. Keating, D. Flynn, R. Aitken, A. Gibons, and K. Shi, *Low Power Methodology Manual: For System on Chip Design*: Springer, 2007.

[5] M. Pedram and J. M. Rabaey, *Power Aware Design Methodologies*. Norwell, MA: Kluwer Academic Publishers, 2002.

[6] H. Esmaeilzadeh, E. Blem, Ren, #233, e. S. Amant, K. Sankaralingam, *et al.*, "Power Limitations and Dark Silicon Challenge the Future of Multicore," *ACM Trans. Comput. Syst.*, vol. 30, pp. 1-27, 2012.

[7] Y. Tsividis and C. McAndrew, *Operation and Modeling of the MOS Transistor*: Oxford Publishers, 2010.

[8] K. Roy and S. C. Prasad, *Low-power CMOS VLSI Circuit Design*: Wiley-Interscience, 2000.

[9] V. Kursun and E. G. Friedman, *Multi-Voltage CMOS Circuit Design*: Wiley Publishers, 2006.

[10] J. E. Stine and J. Grad, "Low Power and High Speed Addition Strategies for VLSI," in *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on*, 2006, pp. 1610-1613.

[11] R. Chau, J. Brask, S. Datta, G. Dewey, M. Doczy, B. Doyle, *et al.*, "Application of high-K gate dielectrics and metal gate electrodes to enable silicon and non-silicon logic nanotechnology," *Microelectron. Eng.*, vol. 80, pp. 1-6, 2005.

[12] IBM, "CMOS 10LPE (CMS10LPE) Technology Design Manual," 2011.

[13] IBM, "CMOS 32SOI (SOI13S0) Technology Design Manual," 2012.

[14] P. R. Groeneveld, "Physical design challenges for billion transistor chips," in *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on*, 2002, pp. 78-83.

[15] J. K. Oustershout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "Magic: A VLSI Layout System," University of California Berkeley1983.

[16] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*: Springer, 1998.

[17] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd Edition ed.: Morgan-Kaufmann Publishers, 2012.

[18] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 5th Edition ed., 2011.

[19] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*: Morgan and Claypool Publishers, 2012.

[20] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, *et al.*, "UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC," in *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, 2007, pp. 22-25.

LIST OF SYMBOLS, ABREVIATIONS AND ACRONYMS

AFRL	Air Force Research Laboratory
ASCII	American Standard Code for Information Exchange
CDS	Cadence Design System
CMOS	Complementary Metal Oxide Semiconductor
DEF	Design Exchange Format
EDA	Electronics Design Automation
EDI	Encounter Digital Implementation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Descriptive Language
ICC	Integrated Circuit Compiler
IPC	Instructions Per Cycle
ISUB	sub-threshold current
ITRS	International Technology Research
LEF	Layout Exchange Format
MIPS	Microprocessor without Interlocked Pipeline Stages
MTCMOS	Multi-Threshold CMOS
NMOS	Negative Metal Oxide Semiconductor
NP	Non-deterministic polynomial
NSF	National Science Foundation
OA	Open Access
PMOS	Positive Metal Oxide Semiconductor
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer level
SoC	System on a Chip
Tcl	Tool command language
TLP	Threads Level Parallelism
VGS	gate-source voltage
VLSI	Very Large Scale Integration
V_T	threshold voltage